

AD-A281 749

1

NAVAL POSTGRADUATE SCHOOL Monterey, California



THESIS

DTIC
ELECTE
JUL 20 1994
S G D

AN APPLICATION OF VIRTUAL PROTOTYPING
TO THE FLIGHT TEST AND EVALUATION
OF AN UNMANNED AIR VEHICLE

by

Mark T. Lagier

March 1994

Thesis Advisor:

Isaac I. Kaminer

Approved for public release; distribution is unlimited

94 7 18 097

94-22634



7918

DTIC QUALITY EMPLOYED 1

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School		6b. OFFICE SYMBOL (if applicable) EC		7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School	
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943			7b. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (if applicable)		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (Include Security Classification) An Application of Virtual Prototyping to the Flight Test and Evaluation of an Unmanned Air Vehicle					
12. PERSONAL AUTHOR(S) Lagier, Mark T.					
13a. TYPE OF REPORT Master's Thesis		13b. TIME COVERED FROM 07/91 TO 12/93		14. DATE OF REPORT (Year, Month, Day) March 1994	
15. PAGE COUNT 79					
16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	Virtual Prototyping, Visual Feedback, Dynamic Modelling, Controller Design Testing, Flight Dynamics Testing, Over the Horizon Piloting		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Virtual Prototyping is an integral part of the ongoing effort to design, test and fly an Unmanned Air vehicle. Current analytical software such as SIMULINK or MATRIXx provide powerful design tools with limited graphical output, that require an intimate knowledge of the underlying dynamic structure. For comprehension, Virtual Prototyping allows an intuitive approach toward understanding the dynamic performance of the model. When the aircraft is flown within visual range of the ground station, a Virtual Prototype display provides the pilot on the ground a close-up view of aircraft response. When the aircraft operates over the horizon, a Virtual Prototype display becomes the only visual link between the pilot and the aircraft. An application of a Virtual Prototype software is presented here with a direct view to implementing the results in the UAV project currently underway at The Naval Postgraduate school.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Isaac I. Kaminer			22b. TELEPHONE (Include Area Code) (408) 656-2972		22c. OFFICE SYMBOL AA/KA

Approved for public release; distribution is unlimited

**An Application of Virtual Prototyping
to the Flight Test and Evaluation of an Unmanned Air Vehicle**

by

Mark T. Lagier
Lieutenant Commander United States Navy
B.S., Oregon State University, 1983

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the


NAVAL POSTGRADUATE SCHOOL


March , 1994

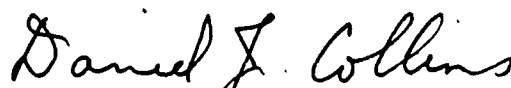
Author:


Mark T. Lagier

Approved by:


Isaac I. Kaminer, Thesis Advisor


Richard M. Howard, Second Reader


Daniel J. Collins, Chairman
Department of Aeronautics and Astronautics

ABSTRACT

Virtual Prototyping is an integral part of the ongoing effort to design, test and fly an Unmanned Air Vehicle. Current analytical software such as SIMULINK or MATRIXX provide powerful design tools with limited graphical output, that require an intimate knowledge of the underlying dynamic structure. For comprehension, Virtual Prototyping allows an intuitive approach toward understanding the dynamic performance of the model. When the aircraft is flown within visual range of the ground station, a Virtual Prototype display provides the pilot on the ground a close-up view of aircraft response. When the aircraft operates over the horizon, a Virtual Prototype display becomes the only visual link between the pilot and the aircraft. An application of a Virtual Prototype software is presented here with a direct view to implementing the results in the UAV project currently underway at the Naval Postgraduate School.

Accession For	
NTIS CRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Dist.ribution /	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION	1
	A. UAV PROJECT	1
	B. DESIGNER'S WORKBENCH	3
II.	GRAPHICS EDITOR	6
	A. GETTING STARTED	6
	B. CONFIGURING THE DWB WORKSPACE	8
	C. CREATING A DATABASE	10
	1. DWB Structure	11
	a. Basic Elements	11
	b. Maneuvering inside the Structure	13
	D. BUILDING A 3-D IMAGE	15
	1. Example — Using the 3-D tools	18
	2. SUMMARY	23
	E. ENHANCING THE MODEL	24
	1. Color	24
	2. Face Reversal	26
	3. Mirror	27
III.	ADVANCED MODELLING	28
	A. CLIP REGIONS	28
	B. PERSPECTIVE REGIONS	31
	1. Utilizing the Perspective Region	35
	a. Scale	36
	b. Aspect Ratio	37

IV.	LINK EDITOR	39
A.	VARIABLES	40
1.	Internal Variables	40
2.	External Variables	40
B.	OBJECT LINKS	41
1.	Translation and Coordinate Links	43
2.	Rotation Links	48
3.	Some Final Notes on Object Linking	50
C.	EYE POINT LINKS	52
D.	PAGE LINKS AND LINKING IN THE PERSPECTIVE REGION	54
V.	COMMUNICATION	55
A.	.VARS FILE	56
B.	DATA FILES	58
C.	SHARED MEMORY AND ETHERNET CONNECTIONS	60
VI.	CONCLUSIONS AND RECOMMENDATIONS	61
A.	CONCLUSIONS	61
B.	RECOMMENDATIONS	61
APPENDIX A	: LIST of PROJECT FILES	63
APPENDIX B	: DATA CONVERSION PROGRAM IN "C"	65
REFERENCES	68
INITIAL DISTRIBUTION LIST	69

LIST OF TABLES

5.1 .VARS FILE STRUCTURE	57
---	-----------

LIST OF FIGURES

1.1	AROD	2
1.2	Archytas	3
2.1	DWB Workspace Environment	8
2.2	Coordinate Input Window	18
3.1	Clip Region Implemented on an a Attitude Gyro	32
4.1	Internal Variables	41
4.2	External Variables in the .vars File	42
4.3	Link Create/Edit Dialogue Box	44
4.4	Coordinate Link Edit Box	46
4.5	Mapping Function Editor	47
5.1	The Comm Editor	56
5.2	The .vars File	57
5.3	The .data File	59

ACKNOWLEDGMENT

I would like to express my appreciation to Professor Isaac Kaminer for his advice and professional counsel without which I could not have completed this work. I would also like to thank Mr. Glen Raudins of Corphaeus Software, Inc. for his patience and input during the development phase of this project. Finally, I would like to thank Mr. Matthew Koebbe of the Computer Science Center Visualization Lab staff for the countless hours he devoted to helping develop a permanent video record of the project.

I. INTRODUCTION

A. UAV PROJECT

The UAV project currently in progress at the Naval Postgraduate School has two major thrusts: Fixed wing research, and VTOL development. The fixed wing research is being conducted on a small aircraft called BLUEBIRD. This is a remotely piloted vehicle with a wingspan of 12 ft, and a payload capacity of approximately 20 lbs. This aircraft is currently being flown at the Naval Postgraduate School, and provides a test bed for communications equipment and as a stable test platform for experimental controllers. There are a number of flight test projects being conducted on the aircraft to refine the dynamic model of the aircraft being used in the controller and communications equipment design.

The second major research area, VTOL development, is being conducted on a platform originally designed by Sandia National Laboratory for the Marine Corps named AROD (see Figure 1.1).

The Aeronautics and Astronautics Department subsequently acquired a number of these aircraft from the defunct program. The AROD was essentially a ducted fan aircraft that was designed to take off vertically, and maneuver principally as a helicopter in hover. It was to remain vertical for the entire flight, and as such, the power requirements and drag penalties induced by this flight mode resulted in unacceptably low range capabilities. The program underway at the Naval Postgraduate School consists of attaching wings to AROD and designing a controller that will allow the aircraft to take off vertically, transition to horizontal flight, and land vertically. This AROD derivative is called Archytas. Figure 1.2 is an artist's conception of Archytas.

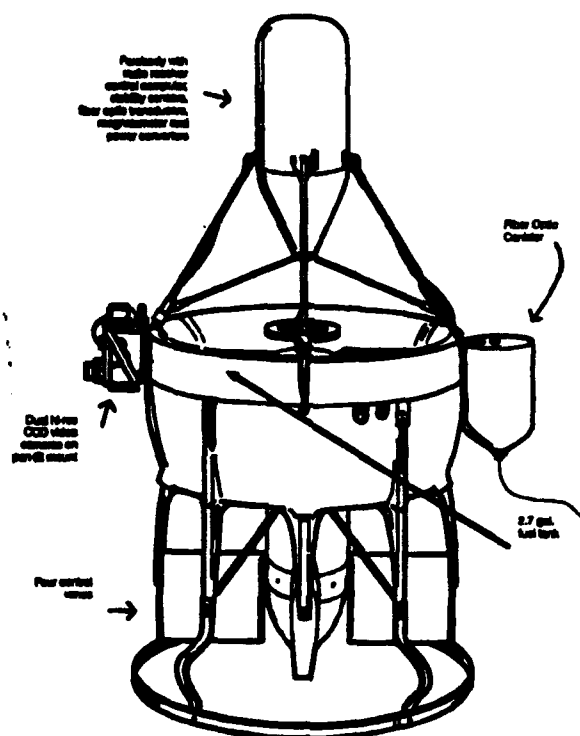


Figure 1.1: AROD

One of the projected missions of Archytas is to provide unmanned over-the-horizon capabilities to the battlefield commander in a package that is man portable or at the most requires the use of a light utility vehicle or pickup. Other features of the projected Archytas mission include an autonomous GPS aided trajectory guidance system, and a future autoland system. There are also provisions for reverting to piloted flight, and it was this requirement along with the over-the horizon mission that drove the need to develop a virtual prototype system. The computational/analytical software existing in the department consists primarily of Matlab and MATRIXx with their associated "graphics" modules, Simulink and Systembuild. Both of these software packages provide extremely powerful methods for designing, testing and analyzing dynamic nonlinear controllers. They are extremely limited, however, in their ability to provide a feedback display suitable for conducting piloted flights of the

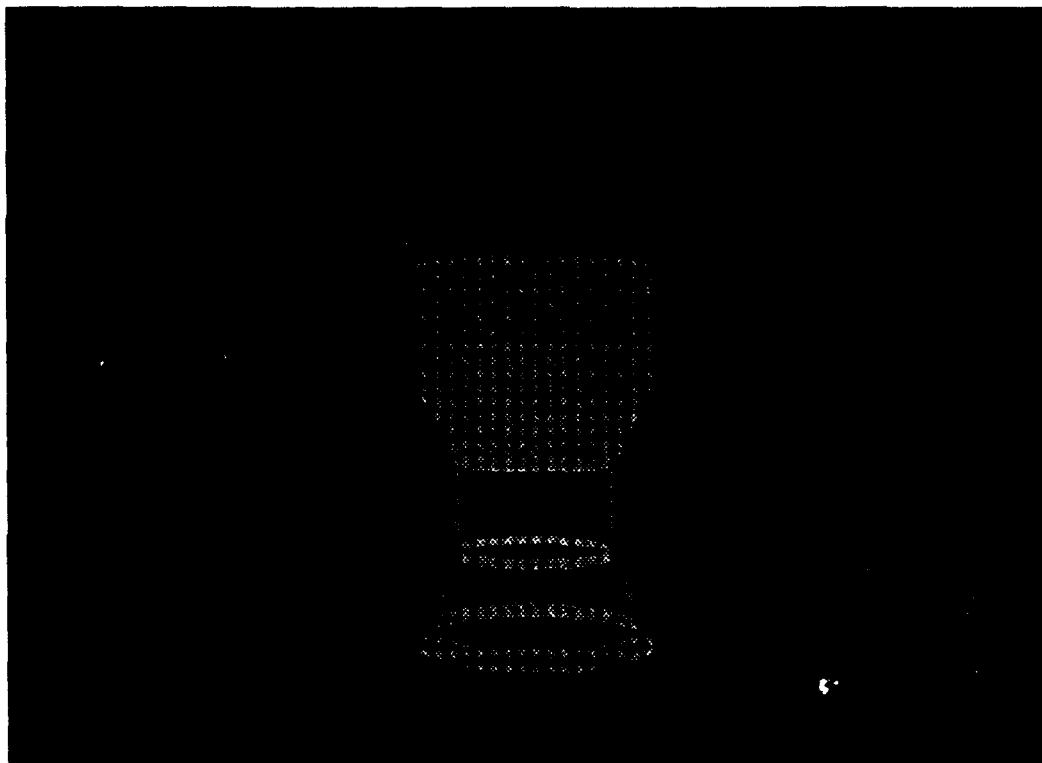


Figure 1.2: Archytas

UAV project aircraft. At best, the simulations run in these software environments can provide a 3-D line plot of the time history of the trajectory, along with a 1-D running plot of selected outputs. It would border on the impossible to expect a pilot to be able to detect and respond to fluctuations in the aircraft attitude utilizing these types of displays.

B. DESIGNER'S WORKBENCH

To fill this shortcoming, the Department of Aeronautics and Astronautics purchased a software package called Designer's Workbench (DWB). DWB is a 3-D inter-graphics program that allows the user to create unique situationally specific models, construct associated cockpit instrumentation and HUD displays, and animate these databases with information derived from an outside source. It provides a means of

supplying instantaneous visual feedback to the pilot, and has become a major contributor to the design and testing of a trajectory controller for Bluebird.

The goal of this thesis was to integrate DWB with other simulation tools, such as Simulink, currently used at the Avionics Lab. To that end, this work is broken into three major categories:

- Creating a database
- Linking
- Communication

Throughout this document, every attempt has been made to provide specific examples of the tasks required to provide a fully functional animation in such a way that the reader is not limited to one application. The software is versatile enough that applications could be expanded well outside of the referenced UAV projects. Attached as Appendices to this document are the following: a list of the files that have been created in conjunction with this project, a description of some of the more complicated links that went into creating a working simulation, and a copy of the C language program used to convert the data saved from Matlab/Simulink simulations to a useable format. In an effort to convey accurately, with the least amount of confusion, each step required to create, link, and animate a database, the following definitions have been used:

- DWB... Designer's Workbench integrated software package.
- Workspace Panel... The area above the drawing window that contains all of the create, edit, and movement icons necessary to manipulate a database.
- Workspace... The drawing environment.

- Structure Chart... The iconized "wiring diagram" of the hierarchy of the database structure. It is invoked by toggling the icon in the extreme upper right corner of the Workspace Panel [Ref. 2, p.3-23].
- Click... The act of selecting an icon in a Graphical User Interface environment. Also select, choose, and open.
- Window... In an open editor (e.g. Mapping, Link, etc.), the area specified for entering or viewing values.

This thesis was not meant to be a replacement for the reference and user's manuals supplied by the manufacturer. The applications discussed in this thesis assume a fundamental understanding of the concepts and mechanics contained in [Ref. 1] [Ref. 2] [Ref. 3] [Ref. 4]. The major thrust of this work was to expand on the mechanics approach of the manufacturer's documentation to allow for greater user applications.

Finally, as with any attempt to portray in writing the procedures necessary to successfully implement a software package, there are times when the vocabulary available for expressing a particular action becomes exhausted. To the greatest extent possible, I have attempted to vary the use of any given word. One such situation that springs immediately to mind, is the act of selecting a button or icon in a Graphical User Interface environment. After *Select*, *click*, and *choose*, the list of available verbs grows very short.

II. GRAPHICS EDITOR

The software utilized to create a Virtual Prototype world for the UAV project is an interactive graphics program called Designers Workbench developed and marketed by Coryphaeus Software of Los Gatos, CA. The software as it exists in the department at this time is composed of two fundamental subgroups. The first of these is the modelling editor, or data base/link editor. This is the environment in which all of the model design, background construction, and link editing is conducted. The second subgroup is the Runtime Module. This is an optimizing program that will automatically execute a previously constructed simulation without the need for invoking the editor.

A. GETTING STARTED

Designer's Workbench has been installed on Indigo3.aa.nps.navy.mil in the Indigo lab administered by the Department of Aeronautics and Astronautics. The installation procedures differ slightly from those listed in the setup manual in that the working files have been installed in `/usr/local/bin` as opposed to the recommended `/usr/cs/bin`. Aside from defining paths differently, this will have no effect on the usability of DWB. Three steps need to be done prior to running DWB on an indigo attached to the aero server.

- The Bitmaps contained in the working files of DWB (`/usr/local/dwb2`) must be linked to the personal account of the user. Let's say the user's account name was Hacker, and Hacker's directory path was `/d4/hacker`. To link the bitmaps, at the command prompt type:

ln -s /usr/local/dwb2/BITMAPS /d4/hacker/BITMAPS

- The environment must be set for the executable file to be able to find the license server and working files. The commands to accomplish this are:

1. *setenv CSL_DIR /usr/local/lic*
2. *setenv CS_DIR /usr/local*

These commands tell the operating system that the license information for DWB can be found in the directory *user/local/lic*, and that the information needed to execute DWB can be found in the directory */user/local*. This can be accomplished either at the beginning of each log-on session, or can be set in the *.cshrc* file of the user. While editing the *.cshrc* file, DWB should be added to the users path. A sample copy of an *.cshrc* is shown in Appendix A.

The last step that needs to be accomplished is to ensure that the license server is running [Ref. 1, p.3-13]. To verify this, type the command:

/usr/local/bin/CSL_server&

If the license server is already running, a response will come back stating that a server is running. If for some reason, the server is not up and running, this command will start it (the *&* sign will allow the server to run in the background).

Once these steps have been executed (normally only needed one time), DWB is ready to run. At the command prompt, type the command "dwb2" and wait for it to initialize and for the graphical editing environment to appear. The remaining sections of this chapter cover the skills required to successfully build a useable model and/or database.

B. CONFIGURING THE DWB WORKSPACE

The initial screen the user faces when DWB comes is shown in Figure 2.1.

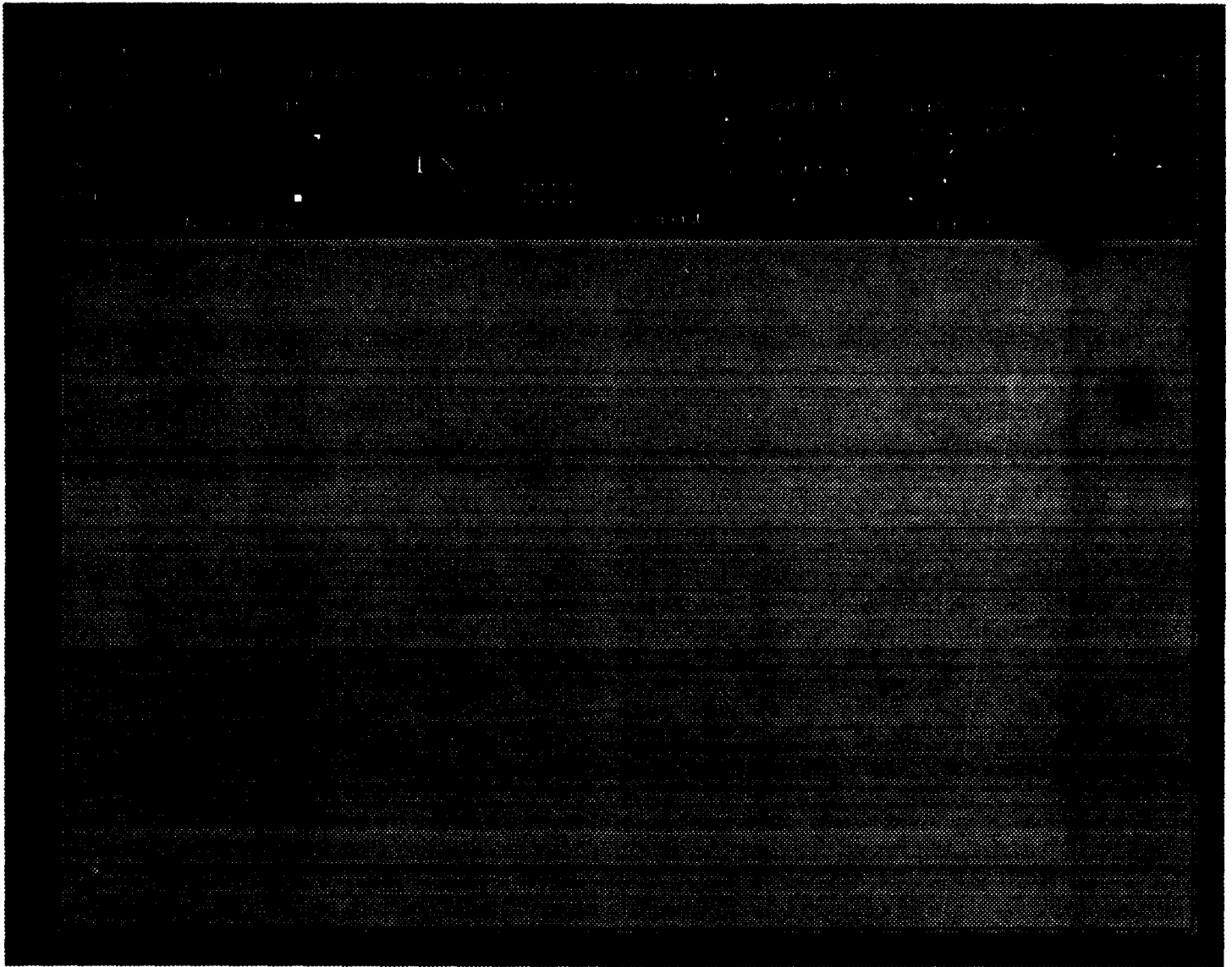


Figure 2.1: DWB Workspace Environment

To open an existing database, the pull down file menu can be used with the graphical interface identical to Windows or Macintosh products [Ref. 2, p.2-2]. To start a new data base, there are several additional tasks that need to be accomplished first in order to maximize the usefulness of our model.

- Decide what scale the database is to be drawn to.

Given the relatively small size of the two primary aircraft associated with the UAV project, drawing an accurately sized version of one of these to import into a full scale airfield will result in either a loss of perspective because you are too close to the aircraft, or a loss of detail because you are far enough away to see the field but lose the detail of the aircraft. As a general rule, the largest item in the database should drive the scale of the remaining items. This is especially important when constructing a cockpit or HUD display. It may seem unnatural to build an airspeed gauge that is, according to your grid size, 300 ft. in diameter, but it will greatly enhance the your ability to use the display. Once you have decided what the largest item in your database is going to be (for the simulations currently running in DWB this is by far the airfield), the workspace needs to be configured as follows:

1. With the original grid on the screen, and nothing selected, go to the edit pulldown menu and select *modify attributes*. A *grid attributes editor* will appear that allows the user to select the units for the grid size (i.e. ft,cm,in,etc.).
2. Click the grid attributes icon on the workspace panel (the one with the question mark on it immediately to the right of the set/pick buttons). In this dialogue box, the size of the grid square can be changed, as well as the spacing of the grid division lines [Ref. 2, p.3-13].

A Word of Caution: If the grid you are planning to create is relatively large, it is a good idea to first change the grid spacing to a manageable number. When DWB initializes, the grid defaults

to a 1 meter square, with a spacing of 8.33 centimeters. If, for example, the size of the grid you were going to create was 1500 ft square, and you changed the grid size without first changing the spacing, the program would attempt to create a 1500 ft grid with major grid divisions spaced 0.0027 ft apart and with 5 minor subdivisions between each of these. The result will be the computer trying to create 2.74×10^6 separate lines in each direction. At best this process will take several minutes, and at worst the computer will lock up.

- Set the preferences.

Under the pull down menu *Misc.*, the third entry will be preferences. Opening the preferences editor will present the user with four settable options. One of the immediate concerns to us is the *auto-write* option. Open the *preferences* pulldown menu in the preferences editor, select the page option, and choose autowrite. This feature determines how many revisions to a drawing or link file need to occur before DWB does an auto-backup on the file. Set the autosave write path to the directory you will be working in. It is also possible to change the number of modifications that take place between autobackups. The auto-saved files will be saved in the specified directory under the extension .xxx.tmp (.xxx being either .dwb or .lnk files.)

C. CREATING A DATABASE

Now that we have configured our workspace, we are in a position to begin construction of a database. Whether it is an airport, airplane or some type of ground vehicle (i.e. tank), the graphics principles remain the same. Make sure you have a

copy of the listed references available as we go through the following example of using the tools available to us in DWB to build a model.

1. DWB Structure

Before we begin the process of building a dynamic model, we need to first examine how DWB will structure the database, how this structuring will affect our model, and how we can modify this structure to suit our needs [Ref. 2, p.3-24].

a. Basic Elements

There are four basic building blocks of the DWB structure:

- Header
- Group
- Face
- Vertex

If this is your first time using DWB (i.e. you have no existing files), go to the directory `/usr/local/dwb2` and open one of the `.dwb` files existing there. Toggle the structure icon (the top one in the upper right corner of the workspace panel) to switch to the structure chart. In this window, you will see at the top of the structure, a white box with the filename and path in it. This is the header box, and it is used to identify the file. Click on the header box, and notice that all of the boxes attached to the header become highlighted by dashed lines. If you toggle back to the workspace, you will find that clicking on the header has selected the entire database. Go back to the structure chart, and click somewhere in the empty part of the window to deselect the header. Observe that attached to the header are a number of red boxes with either a number preceded by a *g*, or a name. These are the groups. They can be considered chapters in the database structure. Once again, click on one of the red group boxes,

and you will notice that all the items in that group become highlighted to indicate that they have been selected in the database. As you can see, selecting a group or an item will also select all of the structure items attached to it. At this point, take note of the small white square in the upper right corner of some of the group boxes. This square indicates that the group has been "compressed", and that there are more structure items hidden below that group. These structure items with attached blocks are called the parent blocks, while the attached items are known as children. With the middle mouse button, click on a red parent box, and you will see it either expand or compress depending on what state it was in before you started. This feature is extremely useful with large databases; it alleviates the necessity of having to page through every structure item to find one that you want. You may also notice that when a "child" group is expanded it may have more children groups attached to it below. It is perfectly acceptable to "nest" groups like this, and is highly recommended as a way of logically organizing a database.

At some point, the nested groups will end, and attached to these innermost child groups you will find a number of multicolored boxes labelled with a figure beginning with *p*. These are the faces (or polygons), and the color of the box corresponds to the color of the face in the database. This feature is convenient when it comes time to group items; you don't necessarily have to worry about finding a collection of faces based on shape, location or the alpha-numeric identifier of an object, if you know what color it is. Note that each face also has a "compressed" indicator square in the upper right corner. If you expand the face, you will find similarly colored blocks with an identifier corresponding to the face block followed by a colon, and several more numbers. These are the vertex addresses. This is the smallest structure item that exists in DWB.

A Point of Technique. Take the time as you proceed with building a database to return occasionally to the structure chart, and reorganize the structure. It is highly recommended that individual items such as gauges, airplanes, buildings, runways, etc. be established in their own group and that you give each such group a meaningful name that will help you keep track of where you are in the database. While it is not critical that this be done during the building phase, it will significantly reduce the time and frustration associated with creating links and running an animation.

b. Maneuvering inside the Structure

Now that we know what the building blocks are, what do we do with them? Close the file that we opened to look at the structure, and start a new file. Don't worry about setting the environment or grid properties, as we will only be here for a moment. Without doing anything else, go to the structure chart. Here you will find a header called *new* with a group labelled *g1* attached to it. Go back to the workspace, and put a few objects on the screen. Make something simple like squares or circles, and when you have created two or three, toggle back to the structure. Notice that all of the items you created have been added to the group *g1*. This is all well and good if you wanted everything you created to be part of one big group, but what if you wanted the next item you created to be separate, or you want to move one of the items already created to a separate group? An example of this would be if you were creating flight instruments for an instrument panel. It would be a good idea to have each of the instruments in a separate group to facilitate linking. (See Chapter IV for more details about linking.)

To put the next item you create into a separate grouping, you need to change the parent. In the upper left corner of the workspace panel, you will see a window labeled *parent*, and in that window will be the group *g1*. Toggling back to

the structure chart, select the header, then go up and click on the window with *g1* in it. The parent label *g1* will change to *new*. Toggle back to the workspace, create another square or circle, and go back to the structure chart. You will notice that a new group has been created (probably labeled *g2* and attached to the header), and the item you just created has been attached to this new group. The Parent box is like a bookmark in the structure. It allows you to selectively choose what you want new objects to be attached to rather than forcing you to find and move them after the fact. But what if you have already created something in one group that you want to attach to another group? This can be done in one of two ways. Let's say that one of the items in the group *g1* actually belongs in group *g2*. Select group *g2* and make it the parent, then select the item you want to move. Go to the *Structure* pulldown menu, select the entry *Attach*. You will see the selected face move to the group *g2*. The other way to move items in the structure is the drag-and-drop method. Clicking on the item you want to move and holding the left mouse button down, drag the face to the group *g2* and when the red *g2* box has a white border around it, let go of the mouse button. The selected face will move to the group *g2*.

As you proceed with the building of your model, the easiest way of keeping track of where you are and where your objects are, is to group your items by logical sorting. For instance, the fuselage and empennage could be in a group titled *aircraft body*, the wings could be a separate group with subgroups of the leading edge, etc. Naming the groups as you go will greatly facilitate editing and linking your model. DWB will allow just about anything in the way of names; there is no restriction on how many letters or numbers you may use, and the software will recognize white space so you can label the groups intelligently as opposed to the standard Unix or DOS conventions of abbreviating to the point of absurdity. For example, a particular group might be labeled:

Instantaneous Vertical Speed Indicator

and would read just like that in the structure chart. To rename a group or face, you can use the Rename menu option in the *Structure* pulldown menu, or a quicker way would be to use the "hot key". This entails selecting the box you want to rename, and then typing the letter "j". The standard dialogue box will appear, and you can proceed as before.

As a final point in this discussion about DWB structure, go up and open the *Structure* pulldown menu. You will see options like "bring to front", "send to back", "send to back one", and "bring to front one" [Ref. 3, p.3-16]. When DWB draws a picture, it must prioritize in some way the order in which the structure is drawn. Unlike the real world, where it is fairly obvious if your view of an object is blocked by something else in your line of sight, DWB has no way of knowing which item should be placed in front of the other, especially when you begin rotating the picture. Selecting the option *Bring to Front* will cause DWB to always draw that object last so that it is always on top, and consequently always visible, even if the database is rotated 180°. Conversely, selecting *send to the back* will cause the item to be drawn first, and therefore on the bottom and covered by everything between the object and the eyepoint. As we will discover later in the chapter, the current level of graphics does not facilitate drawing true 3-D perspective pictures, so along with backface removal and mirroring, changing the order in which things appear in your structure can greatly affect how the finished product looks.

D. BUILDING A 3-D IMAGE

The most important thing to realize about the DWB graphics editor, is that it operates essentially in a 2-D environment. This means that when you create a point, DWB will attempt to default that point to where it thinks it belongs on the

grid. If you attempt to create a point in free space by forcing it off of the grid structure, DWB will make a best guess at what it thinks the third dimension should be, and while it may look satisfactory from the current viewing perspective, it will become immediately apparent when you rotate the structure that the free space point ended up nowhere near where you intended to put it [Ref. 2, p.3-15]. While DWB does have a limited capacity for creating "preformed" 3-D structures, the nature of aerodynamic modelling usually prohibits the use of these items for anything except fundamental subsets. Our first priority then should be to discover how to manipulate the environment to create a 3-D image in a 2-D workspace. Discussed next are some methods of maneuvering the grid orientation to create the third dimension.

- Rotate the grid to an orthogonal plane.

When the workspace starts up, the grid defaults to the X-Y plane. The orientation button on the lower middle portion of the workspace panel allows the user to, among other things, reorient the grid to any of the three principle planes. Simply rotating the grid to one of these planes will leave the center of the grid structure at the (first) initial origin of the picture. This will allow the user to precisely create and orient structures that contain orthogonal members. This is also a quick way to check the scale of the model in different planes.

- Orient the grid to a particular face.

Say for instance, that you have created a cylinder with ten sides. On one of these sides, you wish to place the words *U.S. NAVY*. The easiest and most accurate way to accomplish this is to align the grid to the face that you wish to place the text on. Select the orientation button on the workspace panel, choose the *face* option, and a dialogue box will appear asking the user to select a face. Click the mouse on the face you wish to place the text on, select *OK* on the

dialogue box, and the grid is now attached to the face you selected, with the grid origin located at the centroid of the face. Once the grid has been reoriented, you may change the orientation to a principle plane and still have the grid origin remain at the centroid of the face selected. This can be an extremely useful feature when constructing complex databases (such as the empennage of an aircraft). Note that the same procedures apply to orienting the grid to a particular vertex. The grid will be oriented perpendicular to the normal of the vertex, but the orientation may then be changed as desired. To move the grid back to the initial origin, simply select *origin* under the grid orientation icon.

- **Input the coordinates directly.**

In the extreme upper right corner of the workspace panel, there is an icon labeled $x - y - z$. When activated, a readout of the precise coordinate of the point selected will appear. Contained in this box are also three *delta* windows immediately to the right of the coordinate position as shown in Figure 2.2. Also on the right side of the dialogue box are three buttons labeled freeze $x/y/z$. If you wish to move in a certain plane, freezing one of the axes will allow you to do so. If you need to move along a given line, simply freeze two of the axes [Ref. 2, p.3-19].

A Note of Caution: Be sure to unfreeze the axis before exiting the coordinate input box. Simply closing the box does not unfreeze the frozen axes, and subsequent attempts to create structure items outside of the frozen coordinates will fail.

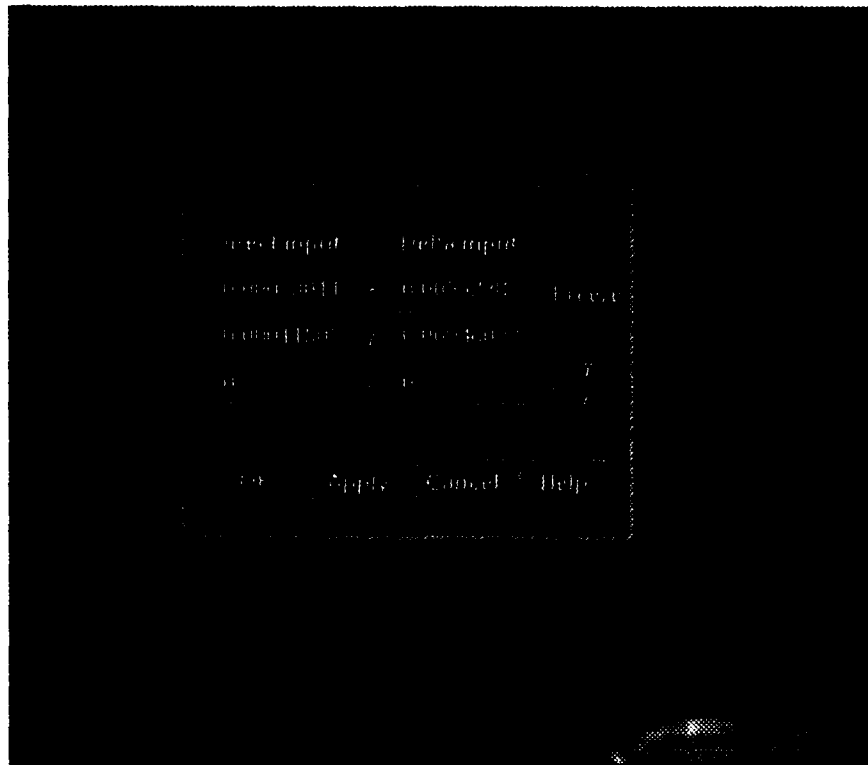


Figure 2.2: Coordinate Input Window

1. Example -- Using the 3-D tools

The following example will demonstrate the concepts previously discussed, and examine other ways of navigating through the 3-D world of DWB. The example will cover, in detail, the steps required to build a 4 ft \times 4 ft. cube by constructing one face, and generating the remainder of the cube through manipulations of replications of this face.

- Open a new file.
- Modify the workspace attributes to dimensions of ft.
- Select the grid attributes (button with the question mark on it), and change the grid division spacing to 1 ft and the grid size in both the x and y planes to be 6 ft so we have some overlap (note the word of caution from the previous

section).

- We now have a 6 ft \times 6 ft grid. Zoom out far enough to see the entire grid and select the square icon from the create menu on the workspace panel (the polygon create icon will accomplish the same thing but will require input of all four of the corners). Place the corners of the square 2 ft. over and 2 ft. up (or down) from the origin of the grid. You should see a white square centered at the origin with sides that are 4 ft long.
- To create a top surface, using the coordinate input method as discussed above, do the following:
 1. Go to the *pick type* menu in the bottom left of the workspace panel, and change the pick type to *face*.
 2. Select the square face you just created by clicking the left mouse button on it. It should be highlighted by a dotted line around it, and a box with a *prxx* symbol should appear in the upper left corner of the workspace.
 3. Select the *duplicate/translate* icon from the workspace panel edit menu (first button in the second row). When the dialogue box appears asking for the duplicate origin, it will automatically default to the centroid of the selected item. Since this point is at the origin of the grid as well as at the center of the square, it will serve as a convenient reference point for placing the top face. Accept this as the duplicate origin. Note that you could have chosen any point on the square as the *duplicate/translate* origin , but it should be something convenient for referencing to and the center of the square fills that requirement nicely.

4. When prompted for the duplicate destination, open the coordinate input window ($x - y - z$ button in the upper right corner), and without entering anything, freeze the x and y axes. Select the *delta z* window, and enter the value -4 . Click the *apply* button.
5. Go back to duplicate destination dialogue box and click *Ok*. Once more back to the coordinate input window to unfreeze x and y , and then cancel the window.

At this point, you have the top and bottom of the box. To generate the next two sides, using the grid rotation features, do the following:

- Select grid orientation in the workspace panel, and orient the grid to the $Y-Z$ plane.
- Now, at the extreme top of the workspace panel, select the *rotate y -90°* button, and you should once again be looking directly down at the grid and should see the two faces created previously at some angle from the side.
- Utilizing the bottom face that was created initially, select the *duplicate/translate* button in the workspace panel edit menu (the first one in the second row). Accept the default origin. When the *duplicate destination* prompt appears, measure along the grid 2 ft. towards the top face and click *OK*.
- At this point, there should be three parallel faces. Select the one in the middle, and click on the rotate icon in the edit menu (second from the left in the top row). Accept the default rotate origin, and rotate one of the axes 90° (it doesn't matter here which one you rotate, since the face is symmetrical). Now there is a vertical face in the center of our box.

- To finish creating the next two sides, you will need to *duplicate/translate* the center face once, and then *translate* it once to put the sides where they belong. First move the grid to the corner of the box where the next side will go. Open the grid orientation window, select *point* and pick one of the corners of the bottom face. You may have to *rot Y* a little one way or the other to be able to see a corner. If you do, select the corner, go back and select the default *x - y* view, and then simply *rot Y -90°*. Go back to the grid orientation window, select the *Y - Z* plane, and the grid is now centered on a corner of what will be the box. At this point, you should again be looking directly down at the grid.
- Reselect the vertical (center) face that you just rotated, and *duplicate/translate* it to the corner of the box the grid was moved to. To facilitate things here, instead of accepting the center of the face as the translate origin, select the corner of the face that will move to the origin of the new grid and answer *Ok* to the *duplicate origin* dialogue box.

A Point of Technique: When you are trying to select a corner or specific point, hold the shift key down while you click on the point with the mouse. This will force the selection point to the closest vertex and will greatly facilitate the ability to match up corners.

- Click on the grid origin to select the *duplicate destination*, and another side of the box is completed.
- Now, move the grid to the opposite side of the "bottom" face, and do the same thing that you just did, except this time use the *translate* button (the first one in the first row of the edit menu) to completely move the vertical face out of the

center of the box. Open the grid orientation window and select *point*. Pick the opposite corner of the bottom face, re-open the orientation window and reselect the $y - z$ plane. Click on the center face to select it again, select the translate button, change the translate origin to the corner of the face corresponding to the grid origin again, answer *OK* to the translate origin, and then click on the grid origin to select the new translate destination.

Go back and look at the structure completed so far before we complete the rest of the cube. Go up to the grid orientation window, select origin, and then select the $x - y$ plane. Reorient the picture to the $x - y$ plane, and you should see what appears to be a single square centered at the origin. Move up to the *rot X* buttons, and click on the up arrow until the box has rotated approximately 90° . You should see a hollow square that appears to be sitting on the grid. Since this is what you wanted to have at this point, that's good. Now to complete the remainder of the sides.

A Point of Technique: By this time it is probably getting a little tough to decipher exactly where the sides begin and end. An important tool in editing is the ability to change the drawstyle [Ref. 2, p.3-6]. Go to the *Graphics* pulldown menu, open the *drawstyle* option, and select *Wire over solid*. The picture will now have solid lines around each face (as opposed to the dashed lines that appear when you select an item), and it is easier to distinguish the faces. While you are here, go back to drawstyle and select the *Wireframe* option. This can also be an useful asset when the drawings get complicated. For the time being, go back to the drawstyle *Wire over solid*.

To create the last two sides, using the rotate about an edge feature in the edit menu, do the following:

- Reorient the picture back to the default $x - y$ plane, select the face that is showing, and select *duplicate/translate*. This time, the duplicated face will remain in position on top of the existing one, so the *duplicate/translate* origin and destination will both be the same. Click *OK* to both boxes.
- Now, *rot X 90°*, and from the edit menu, select the *rotate about an edge* icon (the third one in the first row). When asked for two points, select the two bottom corners of the face closest to you. Click *Ok*, and when the rotate dialogue box appears, rotate the face up 90° . Watch the outline of the face as it rotates. It will be immediately apparent if you are rotating it in the wrong direction.
- Go back to the default $x - y$ orientation, *rot Y 180° (2x +90°)*, and repeat the previous step to complete the last face.

2. SUMMARY

So far, the topics covered have examined some of the basics of creating items and moving them around the workspace to create complex objects (granted the cube is not exactly a complex object, but for the purposes of our discussion it will do nicely). The question that remains is, "is this new structure a functional model from the standpoint of being able to utilize it in a simulation?" To answer this question, go to the *Graphics* pull down menu, and open the function labeled *backface*. You will notice that it has three positions: on, off, and selective. Turn the backface on, and return to the cube. If it looks acceptable from this perspective, start rotating it (either with the workspace panel buttons, or with the middle mouse button). More than likely it will appear that one or more of the sides has disappeared. Obviously,

if this cube is going to be used in a simulation where the backface removal feature will be active, there are going to have to be some changes made to this model. For the time being, turn the backface removal feature off. The following examines some ways to enhance the functionality of the model.

E. ENHANCING THE MODEL

As you look at the finished model of the cube, it should strike you that even with the draw style set to *Wire over Solid*, it is rather difficult to distinguish between individual sides or even determine whether or not you can see all of the sides. This section looks at several features that will significantly improve the ability to modify and utilize this model.

1. Color

Before anything is attempted to fix the problem of the seemingly missing sides of our cube, it would be a good idea to see if something can be done about the fact that all of the sides look the same. The easiest and most practical way to accomplish this is to add some color [Ref. 2, p.3.40]. Open the *properties* pulldown menu, and the first entry you will find is *Color*. This has a submenu, with the following options: color palette, get color, and put color. Select the color palette, and move it out of the way of the drawing. Go back to the cube, and by selecting one face at a time, put a different color on each side. The procedure is very straightforward. Select the color you want in the color palette, select the face or object that you would like to be that color, and go back to the *Properties/Color* menu, and select *put color*. Incidentally, *get color* works exactly the same way in reverse. Once you have colored the six sides, rotate the cube about the different axes and notice that the problem encountered previously when the backface removal feature was active has returned. Because the sides are now of different colors, and distinguishable from one another,

there are certain aspect angles that seem to make it possible to "see through" our cube. There are some colors that should be hidden behind the cube, and some of the colors that should be seen are not there. Before anything else meaningful can be accomplished with the model, this difficulty needs to be resolved.

The problem encountered here is the manner in which a face is created in DWB. Faces are created out of individual polygons, and the software is written in such a way that it will build a face with only one side. If you are looking at the back side of a face with the *backface* option turned off, you are essentially "seeing through" the face to the front side. If you turn the *backface* option on, the backside is no longer "transparent", and you will not be able to see the face.

A Point of Technique: Try experimenting with the backface properties of a face. On an open part of the grid, create a simple square or circle, and rotate it with the *backface* option on and off. Notice that with the item selected and outlined by the dashed line, the outline of the item will still be visible when you view it from the backside. Now, rotate the view back to the front, deselect the item, and check that the *backface* option is turned on. Rotate the face until you are looking at the backside, and try to select it. You will find that you will not be able to select the face. If at some point in time during the course of building a structure you are experiencing difficulties picking a particular face out of a database this may be one of the reasons.

To further illustrate the problem faced here, go back to the *Graphics* pull-down menu, and turn the *backface* option on. Reorient the picture to the default *X - Y* plane, and start rotating the cube. Notice that the different colored faces don't necessarily appear in the order that you think they should. That's because some of the faces were put on the cube facing inward, and you can only see them

when you are essentially looking "through" the cube. When these inward looking faces are turned away from the viewer, you are seeing whatever is on the other side of the cube (provided that the opposing face is also turned inward. Otherwise, it will appear as if both sides are missing and you will see a hole in the cube). Obviously if it was required to maintain a complete sense of perspective while viewing this cube in motion, this situation would be unacceptable.

There are two ways to alleviate this problem. You can reverse the face, or create a mirror image of the face. As the Graphics processing power of the machines in the department increases, some of these type of problems will be overcome by hardware (e.g. hardware *z* buffering). For the time being, let's see if this cube can be made useable with software alone.

2. Face Reversal

The quickest and easiest way to fix the perspective problems with our cube is to individually reverse the faces that are facing inward. To do this, select one of the inward looking faces, (you may have to toggle between *backface* off and on to determine which faces are turned inward), and click on the *Reverse Face* icon in the edit menu [Ref. 3, p.4-3]. Once you have done this with the remaining inwardly turned faces, go back and turn the *backface* option on, and rotate the cube. You will see what you would have expected to see in the first place: a cube with six sides that looks like a cube with six sides. This option is well suited to objects that are composed of a relatively few number of faces, but it can quickly become an exercise in futility when you are trying to pick selected faces out of an engine nacelle that has 200 individual faces. Of course, you can select an entire group or a combination of faces and apply the reverse face option, but the problem remains that the faces that faced properly when you started will now be facing in the opposite direction.

3. Mirror

The next option we have to alleviate the problem of the disappearing faces is to create a mirror image of the face [Ref. 3, p.4-3]. Once you have done this, even with *backface* on, the object will be visible at all times from any aspect angle. Each face must be selected and mirrored individually. This entails selecting each face, clicking the *mirror* icon, and then selecting three points in the plane that you wish to create the mirrored image in. For the cube, this will be a time consuming, but relatively minor, annoyance at worst even with the added workload of reorienting the grid to each face. As you might envision, having to do this for an engine nacelle that has 200 faces will quickly daunt even the most avid computer user.

The acquisition of an upgraded graphics suite by the department will alleviate a number of the problems that currently exist. The addition of hardware *z-buffering* as well 24 bit graphics processors will allow users to do with hardware what has been attempted here with software and brute force alone.

III. ADVANCED MODELLING

The previous chapter covered the skills necessary to construct models that are essentially static, or that fit into the category of "what you see is what you get". DWB has two additional features that will greatly enhance the ability to construct a cockpit display and to combine this "static" display with a "God's eye" or "out of cockpit" view of our simulation. The first of these is the clip region, the second is the perspective region. Both have unique properties and requirements, and will be covered individually with two separate examples.

A. CLIP REGIONS

A clip region is a tool that is used to view a partial segment of a larger structure [Ref. 2, p.4-14]. The two most striking examples of clip region usage would be the creation of a "digital" attitude gyro or, utilizing a linear heading tape for a Heads-Up-Display (HUD). The following example will cover the creation of an attitude gyro using the clip region. For this example, you can either continue in a database you may have previously constructed, or begin with a new file. The procedure is as follows:

- Toggle to the structure and select the header as the parent. This will ensure that each of the items created will be assigned to its own group.
- Starting with an empty portion of grid, create a square and add some color to it. This will be the backplate and border of the attitude gyro.
- In the center of this square, create a sphere with enough sides to give a smooth appearance. (Experience has shown that 30 sides provides a reasonably continuous surface without overdoing it.)

- Change the pick type to *group*, and select the sphere. Go up to the *Select* pulldown menu, and select *isolate*. This will isolate the sphere and expand it to a workable size in the center of the workspace. It will also allow modification of the sphere without altering the backplate.
- Having isolated the sphere, the next thing to do is cut it in half to create upper and lower hemispheres that will become the sky and ground portions of the attitude gyro. Select the *Plane Cut* icon from the edit menu (second one from the left in the second row). The subsequent dialogue box will ask for three cutting points. Carefully select two points on either side of the equator of the sphere, and click *Ok*. The third point will default to the eye point. Select a draw style of *Wireframe*. You should see a continuous cut across the middle of the sphere.
- Go back to the *Select* pulldown menu again, and choose the *Fence within* option. This will force DWB to select only those faces or groups that are completely within the boundaries of the pick square. Change the pick type to *face*, ensure the grid is aligned to the $x - y$ plane, and carefully draw a pick box around the upper hemisphere. You should see all of the faces in the upper hemisphere become highlighted.
- Bring up the color palette, and put a suitable (preferably some shade of blue) color in this highlighted region.
- Select the bottom hemisphere in the same manner, and repeat the last two steps. This time, make the color correspond to the ground (e.g. orange). Now, rotate the sphere and make sure there are no stray faces that did not get colored.

- In the top right corner of the workspace, there will be two buttons. One is labeled *top view*, and the other one will have a label corresponding to the group number of the sphere. Selecting the *top view* will shift the picture back to the original view, and you should see the split color sphere sitting on top of the backplate. If the backplate is in front of the sphere, select it and move it to the back of the structure.
- Go to the structure chart, and with the file header selected as *Parent*, create a new group. Move the two groups you have created so far (backplate and sphere) into this new group and change the name of this new parent to *attitude indicator*. This group will essentially become the "chapter" or parent that contains all of the subsections of the attitude gyro. Once you have moved the backplate and sphere groups into this new group, make *attitude indicator* the parent (in the parent window above) and create another new subgroup. Now move the group containing the sphere into this newest subgroup. The reason for doing this will be covered in more detail in the chapter on linking, but essentially, the group that will become the clip region must be a parent to the group that will eventually be linked. When you get to this point, the main parent *attitude indicator* should consist of three levels of groups: the main level comprised of *attitude indicator*, a level below that containing the backplate and the gyro structure, and the third level below the gyro group that contains the sphere.
- Select the group immediately above the sphere (remember, clip regions must be above linked objects, and the lowest level group containing the sphere will become the link group). Open the Pulldown *Edit* menu, and choose the *Modify Attributes* option. Toggle back to the workspace. When the modify attributes

page appears, notice at the bottom there is a window labeled *Group type* with an entry that says *Normal*. Click on the word "normal", and out of the list that appears, select *Clip Region*. When you have done this, the button immediately to the right of the Group Type window will become active. This is the *Pick Reference Points* button.

- Click on the *Pick Reference Points* button, and a dialogue box will appear asking you to identify the LL (lower left) point. Click somewhere outside the radius of the sphere, but inside the box formed by the outer tangents to the sphere. Click ok on the dialogue box, and the next request will be to select the UR (upper right) point. Select a point diametrically opposed to the first point that you selected and *Ok* the dialogue box.
- Go back to *Modify Attributes* page, and click on the *Apply* option. If you have done everything right, the portion of the sphere outside of the boundaries of the clip region will disappear as if a hole slightly smaller than the sphere was cut in the backplate and the backplate was then moved in front of the sphere. If you don't like the way the clip region looks, repeat the previous steps until you get a region that looks acceptable.

When this sphere is linked to rotate with the aircraft attitude, the clip region will effectively block the view of any portions of the sphere outside of the clip region. It will appear to move much the way a digital attitude gyro moves. Figure 3.1 is an example of a clip region implemented on an attitude gyro.

B. PERSPECTIVE REGIONS

A perspective region is, in a number of respects, similar to the clip region [Ref. 2, p.4-4]. It is a specific portion of the workspace window that can be utilized to provide

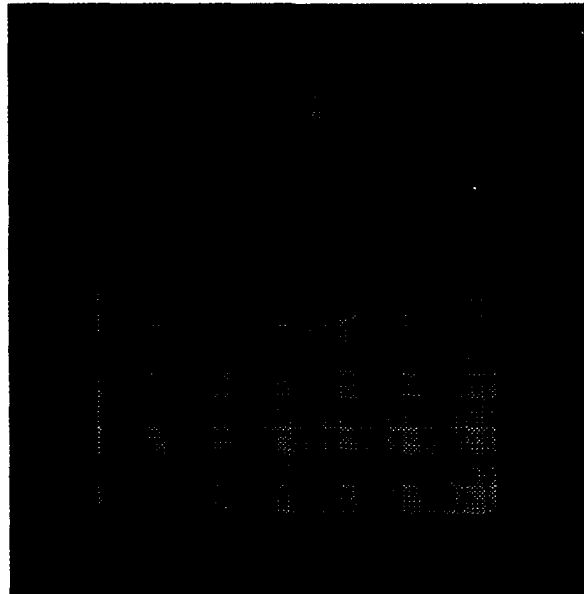


Figure 3.1: Clip Region Implemented on an a Attitude Gyro

a changing frame of reference within that region while the viewing perspective of the area outside of the perspective region remains fixed. The primary difference between the clip region and the perspective region is in the fact that the clip region is applied to an object that is indigenous to the structure (such as the attitude gyro that we created in the last section) while the perspective region is referenced to the entire workspace window. Additionally, to create a perspective region, an external page must be added to our structure. If the workspace area is viewed as the interior of a cockpit, the perspective region provides a window to the "outside world". The option is then available of defining how much window space will be required for a particular application. The procedures entail defining a specific group as a perspective region, defining the area on the screen that will be utilized as the viewing port, and importing a file to act as the viewing model. Before beginning however, the workspace must

be configured so that the perspective region will provide a meaningful presentation. Take the file you just created with the attitude gyro and the clip region, and start from there. The procedures are as follows:

- Open the grid attributes icon, and increase the size of the grid to a point that the attitude gyro created in the last section occupies about a third of the grid width. Zoom out until you can see the complete grid, and translate the entire attitude indicator structure to one side of the grid to provide a clear area in the center of the workspace.
- Change the parent to the file header, and create a new group attached to the header. Toggle to the structure chart, select the group you just created, and open the *Modify attributes* editor. Change the Group type (very similar to what was accomplished with the clip region) to a perspective region.
- Once again, the box labelled *Pick reference Points* will become active. Click on this box, and toggle back to the workspace. The dialogue box will prompt you for a LL (lower left) reference point, and an UR (upper right) reference point. This time however, they will be referenced to the entire DWB workplace viewing area. The size of the area defined will be the size of the viewing window. Pick an area that is roughly square and does not overlap any of the attitude gyro that was moved off to the side. You will notice if you open the *Modify Attributes* box for that group again, that the group type has reverted back to a "normal" group. This occurred because an external page has not yet been attached to the group that was specified as the perspective region, and thus, there is no file identified as the "picture" that will be imported to act as the "outside world".

- At this point, if you created the attitude gyro in the same database as the cube created in Chapter II and are working in that file now, save it and then save it again as a different file name. One will be the working file, and the other will be the import file. Make sure the right path is specified for the *save as* file.
- Reopen the working file and change the parent to the group that was just created and changed to a perspective region. Open the *Structure* pulldown menu, go to the submenu of *Create* and choose the external page option. A file menu box will appear, asking you for the name of the external file to import. Either type in or double click on the file name you saved the import file under, and close the dialogue box. Now if you toggle back to the structure chart, you will notice that attached to the group that you created is a green box with a white header attached to that. The green box is the external page, and the header box is the external file that you just imported.

An Important Note: If changes need to be made in the imported external file, you must open the original file and modify it. The software will not allow you to modify a file while it is in an import or external status.

- Reselect the group that contains the page. Notice that the page and the header box both become highlighted. Toggle back to the graphics screen, and you should see a "window" in the middle of your workspace screen that shows the contents of the file that was identified as your import file. With the same group selected, select the *Modify Attributes* option again, and you will find that the group you identified as a perspective region is now indeed recognized as a perspective region.

1. Utilizing the Perspective Region

The clip region created previously is, for all intents and purposes, ready to be linked and utilized. The perspective region, however, is another matter. Examine the picture in the window of the perspective region. It will appear to be tilted and may either appear as an extreme closeup, or as a very distant view. Toggle back to the structure chart, select the green page box, and open the *Modify Attributes* page. This editor allows modifications to the attributes of the external page, and thus the properties of the viewing area. It has three windows for rotation, three windows for position, a window for field of view, a window for color, a window for scale, a window for aspect ratio, a button for active page, and a window for page number. The software defaults for the rotation windows are: zero for *Y* and *Z*, and 45° for *X*. Change *X* to 0°, and select *Apply*. You should see the view in the perspective region change to an overhead view looking directly down on the grid of the imported database. Note also that the background color (if you can see it) is black. This will be the case regardless of what the original background color was in the imported file. Go to the color window of the *Modify Attributes* page, and enter the color number of the color that you would like the background in the perspective region to be (if you don't know the color number of the color you want you utilize, open the color palette, select the color you want, and note the index number in the lower left corner of the color palette. This is the number that goes in the color box of the *Page attributes* editor). Select the *Apply* button again, and the background color in the perspective region will change.

It is appropriate at this time to mention the fact that with the addition of the perspective region to the database, the possible number of reference coordinate systems in the "world" has increased to three. There is the coordinate system contained in the monitor screen (*x* being positive to the right, *y* being positive up, and

z being positive out of the monitor), the coordinate system of the instrument panel (i.e. the attitude gyro), and finally the coordinate system of the file contained within the perspective region. It is possible to have these three systems orthogonal to each other, and there is nothing technically wrong with that. If at all possible however, you should try to maintain as much coplanarity between the coordinate systems as possible, as this will greatly reduce the bookkeeping workload when the time comes to link the simulation.

As for the remainder of the *Page attributes* editor, you may utilize the position and rotation windows for moving about in the perspective region much the same way you would in the workspace. Ensure that the *Active Page* button is depressed, and check to make sure that the number in the *Page Number* window matches the page number in the green box in the structure chart. This section will conclude with a discussion of two additional features in the *Page attributes* page. These are the scale and aspect ratio features.

a. Scale

The scale option can be useful if the sizing factors between the imported file and the working file in which you created the perspective region are radically different. Let's say, for example, that you created the cockpit file (the working file in which the perspective region was generated) on a grid scale that was $1500ft \times 1500ft$. In creating the import file however, you neglected to change the scale units to feet, and instead, created an object that was 6 inches square. In this case, it would be extremely difficult to utilize this object in the perspective region, because it would be like trying to see a salad plate in a large parking lot. If you were close enough to the salad plate to detect small changes in attitude, you would be too close to the parking lot to be able to tell where you were. If on the other hand, you were far enough back to be able to maintain a relatively good sense of perspective

with respect to the parking lot, you would be too far away from the salad plate to discern any useful information from it. In this case, it would be prudent to change the scale factor in the perspective region to something on the order of 100. This would effectively change the dimensions of the salad plate to 600 inches or 50 ft, and while maybe not completely accurate, it will render the model much more useful.

b. Aspect Ratio

The aspect ratio feature is used primarily for the case in which the perspective region as viewed on the workspace is not square. Try for example changing the reference points of the perspective region to create a window that is twice as long as it is high. Now, go back and look at the picture in the perspective region. It will appear to be stretched and distorted. This will also occur if the file you identify as the import file is not a square field and the perspective region is.

SOME NOTES OF CAUTION ABOUT PERSPECTIVE REGIONS

- Any links that you have built involving a perspective region (i.e. page) will be deactivated when you modify something in the *Page attributes* editor and select either the *Apply* or *Ok* button. If you change the *Page attributes* and accept the changes, you must reload the link file before running the simulation.
- Be careful when using the *Scale* or the *Aspect Ratio* function in the *Page attributes*. If you change the scale of the imported image in the perspective region, this scale change will not be carried over to the data driving the simulation. If, for example, you increase the size of the field by a factor of 2, the position data being input to the simulation will appear to be decreased by half, and it may significantly alter the visual presentation. The same applies to the aspect ratio except

that it gets even trickier. In this case, the imported file proportions may not be scaled by the same amount in each direction, and it can get very messy trying to scale the incoming data to match the aspect ratio changes. If at all possible, try to get the import file and the perspective regions approximately square. It will make the process of linking the simulation much easier.

IV. LINK EDITOR

The previous chapter explored the techniques involved in using a 2-D graphical editor to create a 3-D object that can be used to model or simulate a physical process. In this chapter, the physical process being modelled will be either Bluebird or Archytas in flight. Having constructed an accurate model of either or both of these aircraft, a method must be devised to bring them to life. The process is carried out in the portion of the software called the link editor. This chapter will examine, in detail, the procedures and data necessary to accurately "fly" our aircraft [Ref. 3, p7-1 - 7-43].

The models created in DWB are, for all intents and purposes, "dumb". They cannot determine whether a particular feedback gain is going to be destabilizing, or recognize that the aircraft cannot fly in equilibrium at 90° angle of bank. They serve only as graphical representations of information that is delivered by an outside source. It is this attribute of the software that allows animation links to be applied to any object that we create. For instance, the dynamics of an aircraft could be linked to the cube created in the previous chapter, and an animation developed that could "fly" the cube. Of course the value of the visual feedback would be minimal, but it could be done. With that in mind then, the following discussions relating to link operations will be geared toward the Bluebird and Archytas simulations. The link processes described in this chapter are based on working simulations of both aircraft. The user may choose to use these models, or create an aircraft model of their own. The chapter will conclude with a detailed example of linking Bluebird in a perspective region environment.

A. VARIABLES

Whether the data used to drive the model comes from a data file, shared memory or from an ethernet connection (more on these topics in the next chapter), a standardized way of presenting it to DWB must be defined. The action that will be reproduced on the screen must be driven by some sort of defined variable. For example, generic simulation of simulating a rotating radar beam might use as the driving function, a constant times elapsed time in the azimuth plane, and a sinusoid in elevation plane. For a dynamic simulation that provides interaction, the variables that used to drive the model will have to come from outside DWB's framework and must be created by the user. There are two types of variables available for driving a simulation: internal variables, and external variables.

1. Internal Variables

Internal variables are those variables that have been predefined in the software. A listing of these variables is shown in Figure 4.1.

Note that in addition to the variables listed here, you may also define any combination of these variables as another internal variable and add it to the list. The only stipulation here is that the drivers for the user defined function must be contained in the internal variables listing.

2. External Variables

External variables are the user defined variables that identify the data being presented to DWB from an outside source. These variables are defined in a file accessible to DWB that contains a *.vars* extension in the file name. For a complete discussion of the *.vars* file, refer to Chapter V. A portion of an external variables file is shown in Figure 4.2.

Unlike DOS, the variable names are not limited to 8 characters, and should

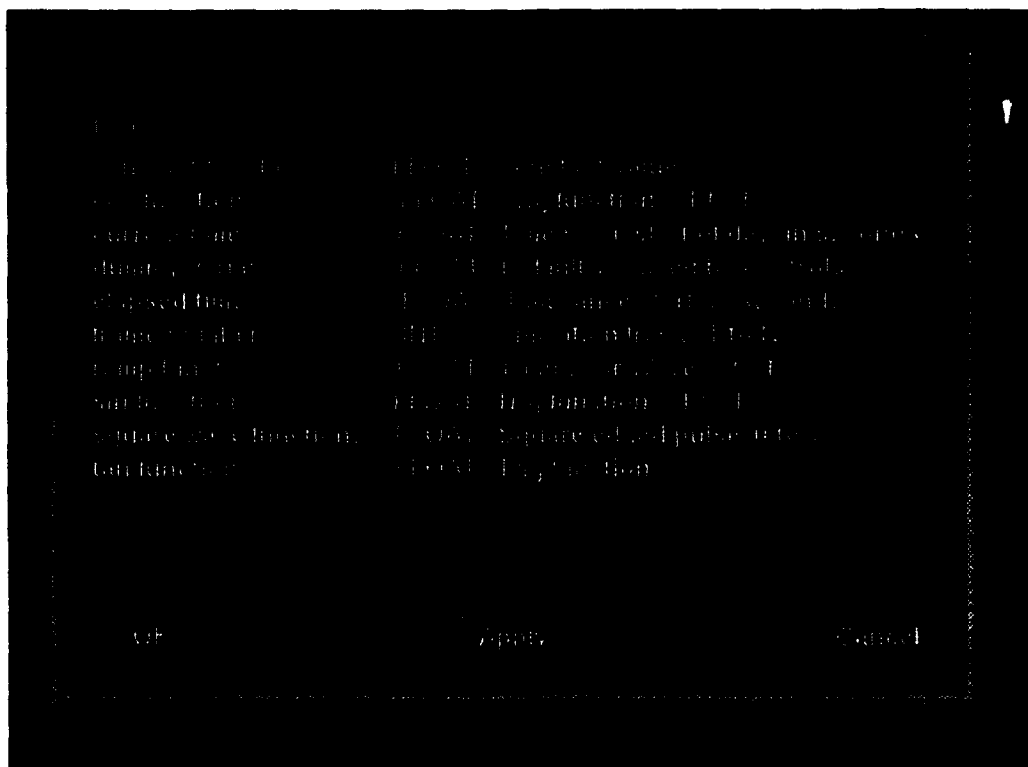


Figure 4.1: Internal Variables

be made as verbose as is necessary to provide a complete description of what the variable contains. Once the *.vars* file is created, it may be used for a wide variety of simulations as long as the format of the data being input does not change. For example, the file *Showtime.vars* may be used to drive the simulation contained in *Perspective5.drt* if the same data format was used for both simulations.

B. OBJECT LINKS

The act of linking an object is what brings the simulation to life. Since the primary application of this software is to facilitate flight visualization of either the Bluebird or the Archytas aircraft, the link discussions in this and the following sections will be focused on aircraft applications. Obviously, the same philosophy would apply to other applications as well.

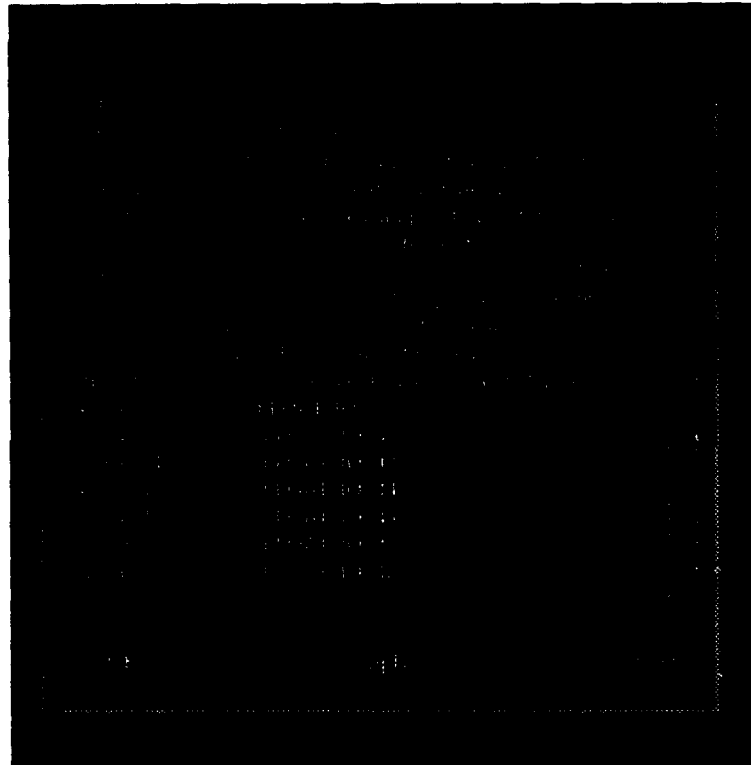


Figure 4.2: External Variables in the .vars File

Using any sort of external information source, be it data file, ethernet, or shared memory to drive the simulation requires that the *.vars* file be loaded. This is accomplished by opening the *Link* pulldown menu, and selecting the *Load Sim Names* option. The standard file dialogue box will appear, and all you need to do is select the appropriate *.vars* file. Once the variable names have been loaded, the creation of links can begin. The first step in the linking process is to select, using the *pick* function, the exact object that you want to link. In the case of an aircraft link, you must select the entire aircraft as a group (each link process will be identified by one object name). It will not work to select the 250 faces that comprise your aircraft group because DWB will not be able to determine which face you want to link. If you want to link a specific item within a group, say for instance, the needle on the airspeed gauge, you can select that particular face if it has only one face, or move all

of the associated faces into a separate group.

A Point of Technique. When linking complex structures that you want to move as a single unit (as would be the case with an aircraft), it is considerably easier to toggle to the structure chart and select the uppermost group in the the structure you want to link. As an example, say you had created a group called *Bluebird*, with attached groups containing the various substructures. If you try to select *Bluebird* out of the workspace, what you will get is the list of the various subgroups. Go back to the structure chart and select the group named *Bluebird*. It will automatically select all of the substructure items below it, and will allow you to create a link called *Bluebird.link*.

Once the object to be linked has been selected, go to the *Links* pulldown menu, and choose the *Create/Edit* option. A blue dialogue box similar to the one shown in Figure 4.3 will appear. If this is the first link for the object, or if you have linked it before but have not loaded that particular link file, there will be one entry listed as shown in Figure 4.3.

DWB will default to a translate link. To change this, click on the *select* button and choose the appropriate link operation. For some of the links, the editing box will appear automatically. For others, you must select the edit button to define the link. The following discussions detail some of the more common linking operations.

1. Translation and Coordinate Links

Translation links accomplish just what the name implies. They allow you to translate an object through some specified path in space. The most common useage of the translation link will be to map external variables to the $x-y-z$ positions of the aircraft as defined by the workspace grid. The coordinate link is simply a grouping of

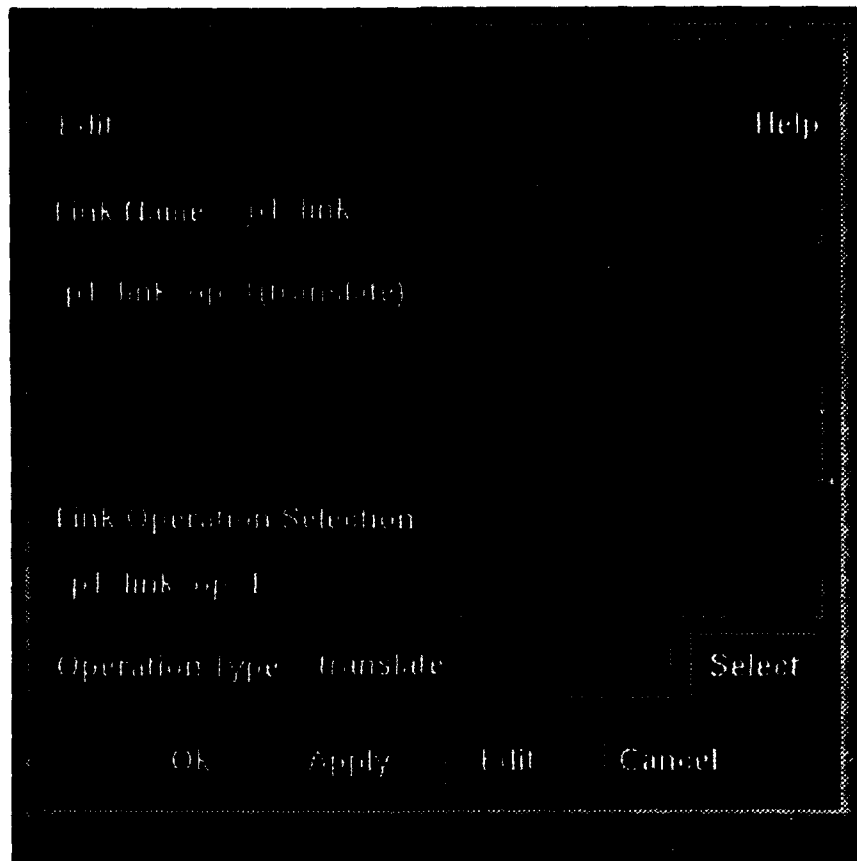


Figure 4.3: Link Create/Edit Dialogue Box

the translation links for the principle axes combined into one editing box. These are used primarily for grouping multiple translation links, and work well as long as the links do not require complex mapping. For most of the applications relating to this project, the coordinate link will be completely satisfactory. For the simulations run to date, the controller model used to generate the data files was an inertial trajectory controller, and it was therefore extremely easy to define as external variables the x, y , and z positions. For other flight models, it may not be as straightforward to generate directly the aircraft inertial position, and this will require some sort of transformation of the u, v , and w states multiplied by elapsed time to provide an inertial position. (Note: u, v , and w vectors define the velocity of the aircraft with respect to the body

frame of the aircraft. By convention, the forward velocity component u is positive out the nose of the aircraft, v is positive out the right wing of the aircraft, and w is positive down.)

At this point in the discussion, it is necessary to reiterate that DWB acts as a "dumb" terminal. The orientation of the linked object with respect to the editor $x - y - z$ grid planes is very important. Let's walk through an example of linking a DWB aircraft to its inertial position as derived from a separate controller simulation. For the purposes of this discussion, assume that the dynamic controller model has the capability of providing the aircraft inertial position as an output. Let us also assume that the inertial x -position, y -position, and z -position have been defined as three external variables in our *.vars* file labeled, respectively, *x_pos*, *y_pos*, and *z_pos*. The scale of the DWB model is such that scaling of the incoming data is not required. Furthermore, assume that the construction of the DWB model was done in such a way that the nose of the aircraft is pointing along the positive y axis of the grid, the right wing is pointing out the positive x axis of the grid, and the positive z axis is out the top of the airplane. Finally, assume that the standard convention of aircraft position labeling has been utilized to generate the data, i.e. x position is positive out the nose, y position is positive out the right wing, and positive z is out of the bottom of the aircraft. The position links will be created using the coordinate link option. Select the entire aircraft (again, ensure that only one group is identified), and open a *Create/Edit* new link box. Select the coordinate link and the editor shown in Figure 4.4 will appear. As you can see, there are three available windows for defining a position mapping.

The windows that have the phrase *frame number* are where the variable names (or the mapped variables) defined in the *.vars* file will go. Click on the window immediately adjacent to the *X Mapped to* phrase, and then move up to the *Select*

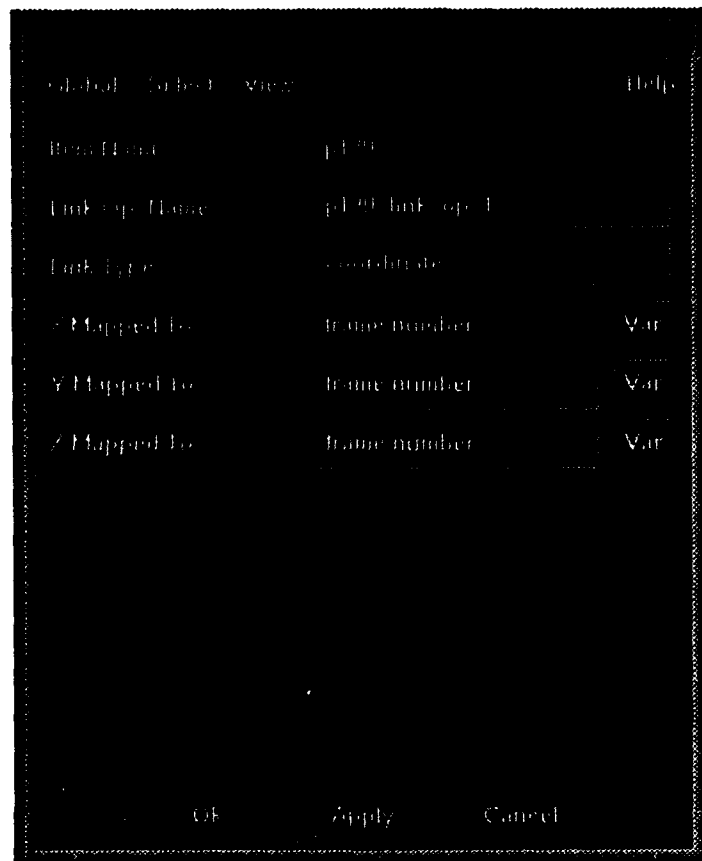


Figure 4.4: Coordinate Link Edit Box

pulldown menu. The first entry is *Variable*, with a submenu to allow you to choose either external or internal variable. Select the external variable option, and a listing of the external variables available in the loaded *.vars* file will appear. Since our aircraft is pointing in the direction of the grid *y* axis, we cannot simply map *x_pos* to the *X* position. We need to map *y_pos* to *X* position for the proper orientation. Similarly, the *Y* position on our grid is going to be driven by *x_pos* from our data file. Finally, the *Z* position will come from *z_pos*, but the two axes are pointing in opposite directions. Once you have entered *z_pos* in the *Z position mapped to* window, go back the *select* pulldown menu, and choose *mapping*. A Mapping function editor similar to the one shown in Figure 4.5 will appear.

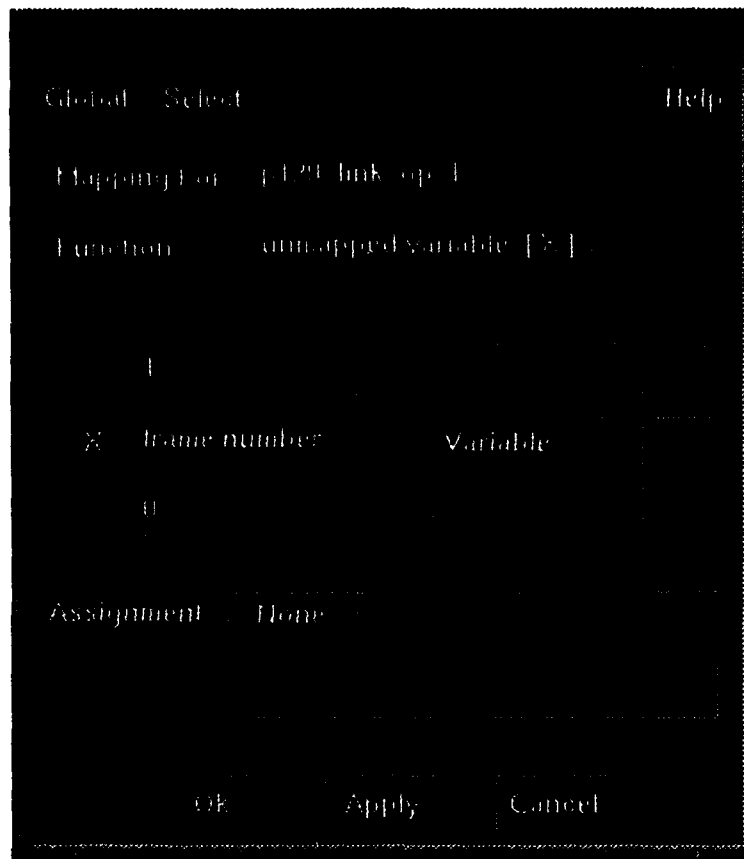


Figure 4.5: Mapping Function Editor

Open the *Select* pulldown menu of the mapping editor, and select the *function* option. A list of available functions will appear, and since for this case, we only want to reverse the direction of our mapping, select *linear*. The linear mapping will provide a means for multiplying the variable by a value (this coefficient value can either be a constant, or can in itself be a mapped function), and/or add another value (here again, the coefficient *C* can either be a constant, or can be a mapped variable). In this case, make the *A* coefficient -1 to change the sign on the incoming data and leave the *C* coefficient unchanged. Click *Ok* for this editor, and aircraft *Z* position is now mapped to the $-z_pos$ external variable. Click on the *Ok* button for the coordinate link editor, and the coordinate link is complete. Utilizing translation links instead if

the coordinate link would have required the same operations and provided the same results. This would, however, necessitate that a separate link operation be defined for each channel of the position data.

A Point of Technique: The *var* button off to the side of the mapping window in the link editor shown in Figure 4.4 is a shortcut button for retrieving a list of variables. If you have a *.vars* file loaded, selecting the var button will retrieve the list of external variables. If there are no external variables loaded, the list of internal variables will appear.

2. Rotation Links

The next type of link to be discussed is a rotation link. Were the animation to be started after creating only the position links, you would see the aircraft moving around the sky in the orientation it started in initially. As long as the aircraft flies only in a straight line, everything will be ok. If, however, the aircraft will be required to turn or maneuver, there must be some sort of rotation applied to the model. As before, an example will be utilized to illustrate the method of linking rotation variables. As previously stated, the assumption will be that Euler (or inertial) angles are available from the data file, and that they have been identified as external variables in our *.vars* file as *phi*, *theta*, and *psi* respectively. As before, ensure the entire aircraft is selected as a group and choose *Create/Edit* from the *Link* pulldown menu. If you are using the same object you used the coordinate link on in the last section, you will see two entries in the box. The second entry will be the default translate link operation, so highlight that one, and change the link type to rotation. The link editor will appear as before (see Figure 4.3). The mechanics of creating the link will be similar to those of the previous section. If the Euler angles are brought in from the data file as radians, you will need to convert these to degrees, which can be done with the linear mapping function as we did with the *Z* position mapping, except the *A*

coefficient will be 57.3 (to convert from radians to degrees). Unlike the translation link, there is no "coordinate" link for rotations. Each rotation link must be defined separately. Let us link ϕ first. Select *phi* out of the external variables file, and apply it to our first rotation link. Now, you must define the plane of rotation for DWB since it has no idea that the variable ϕ is supposed to be a roll about the longitudinal axis. Remembering that the aircraft is oriented with its nose in the positive y direction, a proper ϕ link to the aircraft will be accomplished by choosing *Plane* under the select menu to be the $x - z$ plane. If you were to animate this now, you would notice that the aircraft is banking, but in the wrong direction. This can be corrected in one of two ways. You can either attach a minus sign to the A coefficient or reverse the plane of rotation (accomplished by selecting reverse under *Select/Plane* pulldown menu). The other two rotation channels will be identical except that θ will be mapped to rotation in the $y - z$ plane, and ψ will be mapped to rotation in the $x - y$ plane. Each will be mapped through a linear function with a coefficient of 57.3.

So far, the rotation links created have mapped angles to angles. There is another type of rotation that can be applied to a simulation as well that involves mapping "translation" data to a rotation. The two primary examples of this type of link would be the rotating needles of either an airspeed indicator or an altimeter. In order to create a link of this sort, there must be a defined limit on how far the needle will rotate in representating the data or, in other words, define a mapping between forward airspeed (a translation quantity), and the rotation of a needle on a dial. Let's say for example that we have an altimeter that is good to 10,000 ft. One needle measures the 0 - 1000 ft increments, while the other measures the altitude from 1000 - 10000 ft. The requirement is to link the needles on this altimeter to the altitude of the aircraft, but the aircraft altitude is a "linear" quantity. To accomplish this, we will first select the "1000 ft" needle, create a new rotation link, and map the

link to the variable *z_pos*. Immediately below the *Mapped to* window, there are two windows labeled [*lower mapped limit*] and [*upper mapped limit*]. At the bottom of the page will be two windows labeled *Rotation at min* and *Rotation at max*. For the first needle (our "0 - 1000" ft needle), set the lower mapped limit at zero, and the upper mapped limit at 1000. Rotation at min will be zero, and rotation at max will be 360. For the "0 - 10,000" ft needle, do the same thing. The lower mapped limit will be zero, and the upper mapped limit will be 10,000 ft. The rotation at min will be zero, and the rotation at max will be 360. With this setup and with the aircraft at 5000 ft, the first needle will be straight up (rotation at max is 360), and the second needle will be pointing straight down, since it is halfway between the lower and upper mapped limits, and will have rotated 180°.

If you noticed, there was no need to select a rotation point or origin when we created the rotation links for the Euler angles of the aircraft. When a rotation link is created, DWB will default the center of rotation to the centroid of the object, and in the case of the Euler angle rotations, this was exactly what was desired since the centroid of the model closely approximates the center of gravity of the aircraft. A rotating dial indicator however, needs to have as its center of rotation the base of the needle. While you still have the link editor open, go to the *Select* pulldown menu, and choose the *Origin* option. A dialogue box will appear asking for the new origin or point of rotation. At this time, select the base of the needle, close the *select origin* dialogue box, and accept the links you have created. The link file now consists of links to the three positions and three rotation angles of the aircraft, and links to the aircraft altitude represented on a rotating dial indicator that can be used in a cockpit environment.

3. Some Final Notes on Object Linking

Next, several lessons learned using object linking are discussed.

- **Translation links must come before rotation links.** DWB will process the the links in the link file in the order in which they are listed. As long as the translations get executed first, the simulation will be slaved to the original grid. That is, the animation routine will translate the object through the original frame of reference. If however, you placed a rotation link prior to the execution of a translate, the reference coordinate system will be changed by the rotation links. What will happen is that the object will still translate along the original coordinate frame, but the rotation links will not remain fixed to the same reference frame. This will cause your linked object to essentially "tumble" through space.
- **Rotation Link Order** The normal order of rotation for a body to inertial rotation is 3 - 2 - 1 ($\psi - \theta - \phi$). For some reason, ordering the rotation links in DWB seems to work best in the 2 - 3 - 1 ($\theta - \psi - \phi$) order. If you cannot get your linked object to respond properly, try changing the order of the rotation links.
- **Clip Region Links.** In order to successfully link a clip region, (i.e. if you are going to make an attitude gyro), the structure of the major group that contains the sphere must look like this:

Clip Region Group

Link Group

In other words, create a sphere, color it, and attach all of the pieces to a group (for purposes of illustration, this group will be called the link group). Now, take

that group and attach it to a group one level above it, this one called the clip region group. The result of this structuring is that the link operations do not "know" that a clip region exists, and the entire sphere will move. If the clip and link groups are one in the same, or the order is reversed in the structuring, the link will attempt to move the clip region, with unsatisfactory results (it will appear that the gyro is sliding off of the backplate).

C. EYE POINT LINKS

Eye point links are extremely useful from a simulation design standpoint. Rather than attempt to identify a single fixed point in space from which to view an entire simulation, they provide the flexibility to move with the animation. They also provide a relatively simple method for viewing a simulation, as they can be used without adding the complexity of external pages and perspective regions. If you are building a database with a model that will eventually be imported into a full scale perspective region, the eyepoint feature is a good way to test the links and external data as you go. To create an eyepoint link (in a non - perspective region environment), the *file header* must be the link object. Eyepoint links are fundamentally different from the regular object links in that the frame of reference used for the eyepoint is always fixed to the monitor screen (or eye port). Recall that the object links that discussed in the last section were grid dependent. That is, if you changed the orientation of the database with respect to the grid, the link orientation would also change. With eyepoint links, the monitor is always the $x - y$ plane, and z is always positive out of the screen toward the viewer (x is positive to the right, and y is positive up). Therefore, the eyepoint link mapping can be very different than the object link mapping.

Eyepoint links will be created in exactly the same way that translation links were defined for the aircraft. The eyepoint $x - y - z$ positions must be defined with

respect to the external variables, as well as any rotations that may be desired for the eyepoint. Once the eyepoint links have been defined, it will be necessary to determine a starting position for the eyepoint. The default starting point is the grid origin. If the objective of a simulation was to follow an aircraft through a series of maneuvers while in flight, it may be best to link both the eyepoint and the aircraft position to the $x - y - z$ position external variables. Now, if you accepted the default eyepoint starting position, and also had the aircraft starting at the grid origin, you would not see the aircraft during the simulation because the eyepoint would be right on top of it. To counter this, select the origin to be a point somewhere behind the starting point of the aircraft. The other option would of course be to move the starting point of the aircraft by actually translating it in the database, but this can get cumbersome, especially if you are dealing with a perspective region, and the associated external files.

As a final note about eyepoint linking, it is pertinent to point that the eyepoint initial position could have also been changed through use of the mapping editor. This method does have its drawbacks, and can get extremely complicated. Let's say, for example, that you had linked the eyepoint and the aircraft to the same translation variables, and had linked the eyepoint to rotate in such a manner that it would always be looking at the aircraft from behind. If, instead of changing the initial position of the eyepoint, you simply mapped the eyepoint $x - position$ to the aircraft $x - position$ minus 200 ft, the eyepoint would indeed jump to 200 ft behind the aircraft when the simulation started. If the aircraft were to turn 180° however, the eyepoint would now be "in front" of the aircraft and looking in the wrong direction. This is not to say that this type of eyepoint position linking could not be accomplished, but it does require a degree of sophisticated linking.

D. PAGE LINKS AND LINKING IN THE PERSPECTIVE REGION

Page links are the tool that we will use to link the eyepoint when we are utilizing a perspective region. If we were to link the header of the working file as was done in the previous section, it would effectively leave the viewer static with respect to the perspective region, and it would create a sense of moving about the cockpit. Aside from the link object however, the mechanics for creating an eyepoint link in the perspective region are much the same as those discussed in the previous section. To create an eyepoint link in a perspective region, you will need to link the external page to which the imported file is attached. Select the external page box in the structure chart, and *Create/Edit* an eyepoint link for it in the same manner as the eyepoint created in the last section. The most significant complication that can arise with an eyepoint link in a perspective region is when the coordinate systems for the databases are different. Recall from Chapter III that, by the time you get a perspective region and cockpit display constructed, it is possible for the eyepoint, instrument panel, and the airfield coordinate systems to be orthogonal. This can significantly increase the complexity of the links in the simulation. If you do end up with some degree of orthogonality in the databases, remember that the eyepoint coordinate system is always based around the $x - y$ plane of the monitor screen and the variables or positions you link the eyepoint movements to will be referenced to the coordinate system of the file imported into the perspective region.

V. COMMUNICATION

This leaves us now in a position to discuss the options available for driving a simulation from an outside source [Ref. 4, sec.7]. DWB supports the following communication methods:

- Data Files
 1. ASCII data
 2. Binary data
- Shared Memory
- Ethernet
 1. UDP ethernet
 2. TCP ethernet

Each of these forms of communication uses a slightly different format for presenting data to the DWB animation module. To configure the communications for a simulation, the comms editor must be selected through the *configure comms* option of the *Animation* pulldown menu. Figure 5.1 shows the comms editor.

It is here that the method of communicating with DWB is defined. The set up procedures for the different types of communications vary slightly, but all of them consist essentially of telling the software where the information is coming from, and how the information is identified. There is one point of commonality between the communication types, and that is that each must have access to a *.vars* file for the data to be useful. This will be examined first.



Figure 5.1: The Comm Editor

A. .VARS FILE

The *.vars* file can be thought of as the phone book for the animation module [Ref. 4, p.4.2]. In it, the user defines all of the variables being presented to the simulation. It is here that the relative position of each variable in the incoming stream is defined, as well as the format of that variable. A portion of the *.vars* file used for the Bluebird animations is shown in Figure 5.2.

The *.vars* file contains two critical pieces of information about the variables. First, what type of number the variable is, and second, where the variable is in the data field. As will be examined in the following section, the *.data* file allows the use of up to 50 variables. The *.vars* file, however, gives us the capability of defining up to 240 variables in varying formats. Table 5.1 lists the breakdown of the number of each type of variable allowed when driving a simulation with an ASCII data file. For all

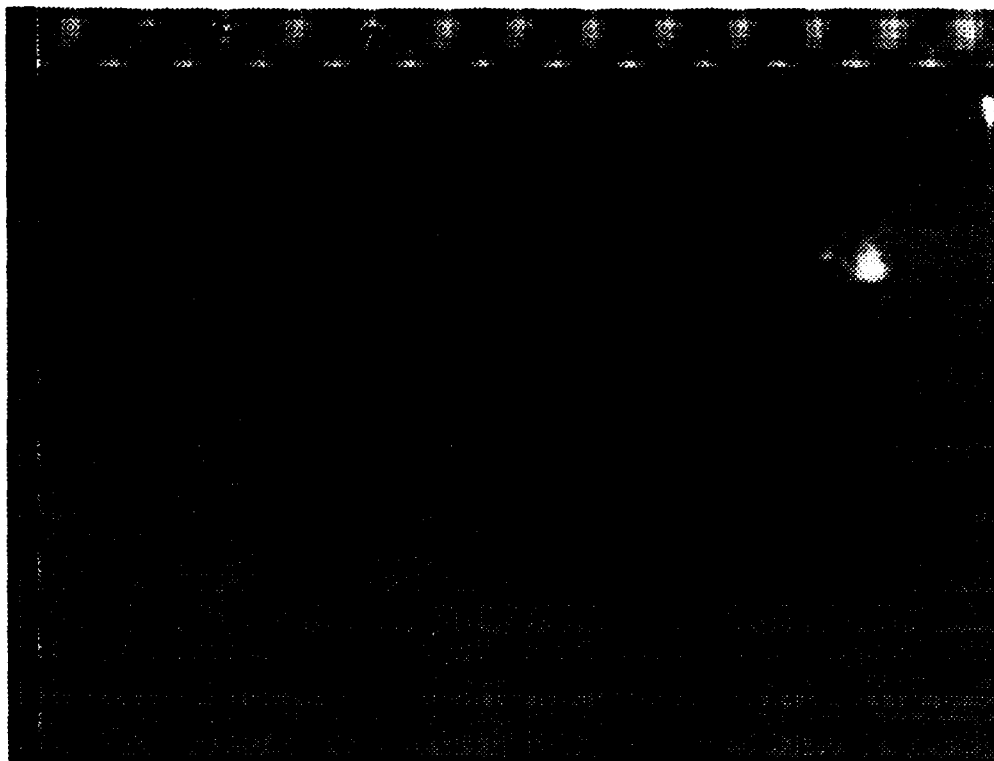


Figure 5.2: The .vars File

other types of communication, there is no limit on the number of variables that may be defined.

TABLE 5.1: .VARS FILE STRUCTURE

Floating point variables	150
Integer variables	50
Character strings	40

The three variable types listed in Table 5.1 are the only types of variables allowed in the DWB environment, and for simulations being driven by ASCII data files, the *.vars* must contain variables within the constraints of the values in Table 5.1. Again, for any other type of information driver, the allowable number of definable variables is not limited. For the purpose of the Bluebird simulation, the data file consisted of

25 states and their values for a specified period of time, so the *.vars* file used 25 of the 150 available floating point variables. The names of the variables can be modified to suit the individual application; the only constraint here is that the naming convention must conform to standard Unix format. If for example, in addition to the variables identified in Figure 5.2, we wished to add the variables true airspeed, groundspeed, and engine rpm, that portion of the *.vars* file might look like:

```
FLOAT      true_a/s      (comment about the variable)
FLOAT      ground_spd    (comment)
FLOAT      engine_rpm    (comment)
```

Once the variables have been defined, you will not need to type them in every time. Rather, save it as a *.vars* file (for instance: showtime.vars).

A Point of Technique: Rather than typing out 240 lines of a *.vars* file, it is much easier to copy an existing file and modify it accordingly.

B. DATA FILES

Driving a simulation with a data file provides a means of running the simulation as a stand alone process [Ref. 4, p.4-7]. We need not have any other communication interfaces active, and it affords us the flexibility of reviewing past simulations, or conducting demonstrations. While a data file does not provide real time capability, it can be a valuable analysis tool. The format of the data file is quite different from that of the *.vars* file. Whereas the *.vars* is a columnar array of 240 fields, the data file is a row array of 51 fields. That is to say that the data file must contain 51 columns (fields) of incoming data. If the simulation does not require 51 variables, the remaining fields must be zero filled. Within the data file, the first 40 fields after the integer time step must be of a floating point format, and the last 10 columns are reserved for string variables (i.e. character strings). The number of data points in each field (i.e. the

length of the columns under each row) is immaterial, but the number of fields is fixed. Additionally, the first column in the data file must be a set of sequential integers. Figure 5.3 is a portion of the data file used in the Bluebird simulations. While it is possible to configure DWB in such a manner that it can accept binary data, there are currently a number of bugs in that portion of the software, and as such it is best to stick to the ASCII data files.

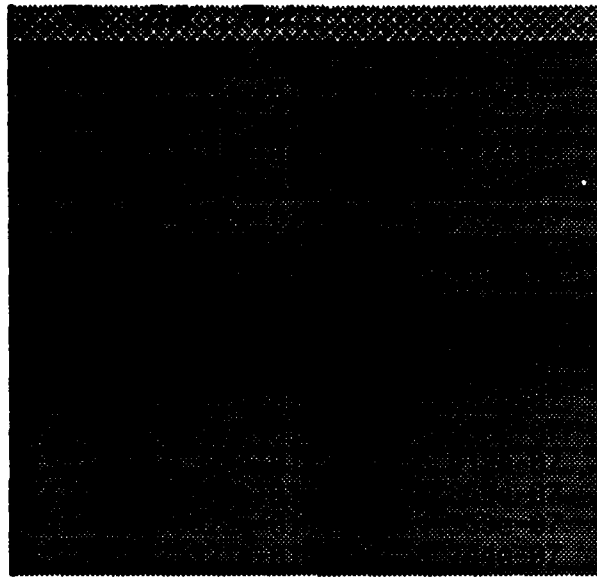


Figure 5.3: The .data File

All of the virtual prototyping conducted in the Department to date has been driven by data obtained from dynamic simulations run in Matlab/Simulink [Ref. 5]. The data was obtained by saving a sampling of each state to a data file in the Matlab workspace, filling the unused portions of a $ffff \times 51$ matrix with zeros, and then saving this matrix in ASCII format. Since it is not possible in Matlab to save a number in anything but floating point format, it became necessary to process the file through an executable C code that converted the first field to a time step integer. This converted data file was then delivered to DWB to drive the simulation. The complete details for doing this conversion are contained in Appendix APPENDIX B.

Through a process of trial and error, it was determined that a significant slowdown in the animation occurred when the sampling rate out of the Simulink simulation was greater than about 10 Hz.

C. SHARED MEMORY AND ETHERNET CONNECTIONS

Shared Memory and Ethernet communication methods are the two primary methods for DWB to communicate to the outside world [Ref. 4, p.4-10]. At this time, the hardware and procedures for achieving a real time link to DWB are being developed. The documentation in [Ref. 4, chap 7] contains a fairly indepth discussion of these methods of communicating, and as such that discussion will not be repeated here. The implementation of either method will require a considerable knowledge of the Unix/Network system.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The addition of DWB to the repertoire of analytical and computational software in the department has provided a major boost in the realization of flying a VTOL/transition autonomous UAV. Having graphical visual feedback available to the pilot significantly increases the chances for completing a successful mission. The software has also become an integral part of the controller design process as an analytical tool, and with the addition of the ISI AC-100 processor, it will also be an integral part of the hardware-in-the-loop testing of various components before the flight test phase. The software as currently installed is for the most part extremely user friendly, and although a number of shortcomings and bugs were discovered during the implementation process, the updated version of the software scheduled to be released in Mar '94 will correct the vast majority of these.

B. RECOMMENDATIONS

The software was designed to be run on SGI machines that had at least 32 MBytes of RAM, and 24 bit graphical processing capability. The SGI machines installed in the Avionics lab at this time have 16 MBytes of RAM, and only 8 bit graphical processing capability. To be able to utilize the software to its full extent, the hardware needs to be upgraded. Aside from hardware upgrades that will significantly improve the performance characteristics of the software, it should be noted that the preliminary development of the virtual prototype models for this project was conducted on release version 2.0 of DWB. A number of bugs and utilization problems

were discovered during this preliminary implementation. Rather than recount the difficulties encountered, it will be sufficient to note that release version 2.1 is scheduled for shipping in March of 1994, and initial reports indicate that the performance of the software has been improved by an order of magnitude.

APPENDIX A: LIST of PROJECT FILES

All of the files that were generated during the course of this project have been transferred to Professor I. Kaminer. The following is a listing of the files containing the final version of several of the projects associated with this work:

- **Perspective5a.xxx.**

Perspective5a (P5a) is a collection of files that were compiled to run a simulation of Bluebird in flight driven by data taken from a Simulink simulation of a trajectory controller. P5a was implemented using a perspective region to provide an out of cockpit view, with 5 cockpit instruments linked to the simulation. The aircraft Bluebird is linked to the "actual" position of the aircraft, and the eyepoint is linked to the commanded trajectory position to provide a means of determine variations in the aircraft position from the commanded position. P5a is found in */home/lagier/cs_world* and has associated *.lnk*, *.drt*, and *.cco* files. The *.vars* file used with simulation can be either *showtime.vars* or *perspective4.vars*.

- **Archyt_show.xxx.**

Archyt_show is a collection of files compiled to provide a fictional rendering of Archytas in flight. The file demonstrates the six degrees of freedom available from the controller, and the VTOL/transition characteristics of Archytas. At the time of this writing, there was no data available to construct a data file showing the dynamics of Archytas. This would be a good follow on project.

- **Showtime/Showtime2.xxx**

The Showtime files were the first full animation files run during this project.

There is no perspective region or accompanying cockpit instrumentation, but the degree of resolution is better in these simulations than the perspective series due primarily to the fact that there is a larger viewing area. If you want to quickly test a link or data file, these files are well suited to that task.

- **Importn.dwb**

The import series of files were replications of Monterey.dwb used to import into the perspective regions of the perspective series. P5a uses it's import file import5.dwb. This is the database you must change if you want to make changes in the P5a viewing area.

The rest of the files in the directory are files that were used as stepping stones to get to the files listed above. They are, for the most part, either incomplete or nonfunctional for one reason or another.

APPENDIX B: DATA CONVERSION PROGRAM IN "C"

This is a reproduction of the C language program that currently resides on the PC-486 in the Avionics lab. The purpose of the program is to take as input a matrix of *ffffx51* floating point ASCII numbers, and rewrite the first column into an integer format.

```
#include <stdio.h>
```

```
#include <math.h>
```

```
void main(void)
```

```
{
```

```
float num;
```

```
int n;
```

```
int m;
```

```
int length;
```

```
int start;
```

```
char namein[20];
```

```
char nameout[20];
```

```
FILE *fp1,*fp2;
```

```

printf("This program takes a MATLAB data file saved in ASCII format\n");
printf("and changes the first column's data structure to integer format\n");
printf("in order to make it compatable as an input file to Designer's Workbench.\n'

printf("Please enter the file name of the MATLAB data file (ascii format).\n\n");
scanf("%s",namein);
printf("\n");
printf("Please enter the file name of the DWB format file.\n\n");
scanf("%s",nameout);
printf("\n");
printf("How many rows (time steps) are in %s ?\n\n",namein);
scanf("%i",&length);
printf("\n");

fp1 = fopen(namein, "r");
fp2 = fopen(nameout,

for ( n=1; n<length; n++)
{
for( m=1; m<52; m++)
{
fscanf(fp1,"%f",&num);

if( m==1 )
{

```

```
fprintf(fp2, "\n%i\t", n);  
}  
else  
{  
    fprintf(fp2, "%6.3e\t", num);  
}  
}  
}  
  
fclose(fp1);  
fclose(fp2);  
}
```

REFERENCES

1. Coryphaeus Software, Inc., "License Manager User's Guide" June 1993.
2. Coryphaeus Software, Inc., "Designer's Workbench *Users* Manual" June 1993.
3. Coryphaeus Software, Inc., "Designer's Workbench *Reference* Manual" June 1993.
4. Coryphaeus Software, Inc., "Designer's Workbench *Link Editor/Run Time Module User's* Manual June 1993.
5. Hallberg, E. N. , *Design of a GPS Aided Guidance, Navigation, and Control System for Trajectory Control of an Air Vehicle*, MSAE Thesis, Dept. of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA, March 1994.

INITIAL DISTRIBUTION LIST

	No. of Copies
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Dr. Isaac I. Kaminer Department of Aeronautics and Astronautics, Code AA/KA Naval Postgraduate School Monterey, CA 93943-5000	5
4. Dr Richard M. Howard Department of Aeronautics and Astronautics, Code AA/HO Naval Postgraduate School Monterey, CA 93943-5000	1
5. Chairman Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, CA 93943-5000	2
6. Thomas F. Lagier 73490 Siesta Trail Palm Desert, CA 92260	1

No. of Copies

7. LCDR Mark T. Lagier
TACTICAL AIR CONTROL SQUADRON TWENTY-TWO
FPO NEW YORK 09501-6542

1